Next-Generation Technologies Assignment 5- Diffie-Hellman

Problem 1 output and code:

Text

Description automatically generated

Text

Description automatically generated

Keys class:

class Keys{  
  
private static long power(long a, long b, long p)  
 {  
 if (b == 1)  
 return a;  
 else  
 return (((long)Math.*pow*(a, b)) % p);  
 }  
  
// Driver code  
public static void main(String[] args)  
 {  
 long P, G, x, a, y, b, ga, gb;  
  
 // Both the persons will be agreed upon the  
 // public keys G and P  
  
 // A prime number P is taken  
 P = 13;  
 System.*out*.println("The value of P:" + P);  
  
 // A primitive root for P, G is taken  
 G = 6;  
 System.*out*.println("The value of G:" + G);  
  
 // Alice will choose the private key a  
 // a is the chosen private key  
 a = 5;  
 System.*out*.println("The private key a for Alice:" + a);  
  
 // Gets the generated key  
 x = *power*(G, a, P);  
  
 // Bob will choose the private key b  
 // b is the chosen private key  
 b = 2;  
 System.*out*.println("The private key b for Bob:" + b);  
  
 // Gets the generated key  
 y = *power*(G, b, P);  
  
 // Generating the secret key after the exchange  
 // of keys  
 ga = *power*(y, a, P); // Secret key for Alice  
 gb = *power*(x, b, P); // Secret key for Bob  
  
  
 System.*out*.println("Secret key for the Alice is:" + ga);  
 System.*out*.println("Secret key for the Bob is:" + gb);  
 }  
 }

Primitive class:

import java.util.\*;  
  
class Primitive  
{  
  
 // Returns true if n is prime  
 static boolean isPrime(int n)  
 {  
 // Corner cases  
 if (n <= 1)  
 {  
 return false;  
 }  
 if (n <= 3)  
 {  
 return true;  
 }  
  
 // This is verified so that the below loop can skip the middle five numbers.   
 if (n % 2 == 0 || n % 3 == 0)  
 {  
 return false;  
 }  
  
 for (int i = 5; i \* i <= n; i = i + 6)  
 {  
 if (n % i == 0 || n % (i + 2) == 0)  
 {  
 return false;  
 }  
 }  
  
 return true;  
 }  
  
   
 static int power(int x, int y, int p)  
 {  
 int res = 1; // Initialize result  
  
 x = x % p; // Update x if it is more than or equal to p  
  
  
 while (y > 0)  
 {  
 // If y is odd, multiply x with result  
 if (y % 2 == 1)  
 {  
 res = (res \* x) % p;  
 }  
  
 // y must be even now  
 y = y >> 1; // y = y/2  
 x = (x \* x) % p;  
 }  
 return res;  
 }  
  
 // function to store prime factors of a number  
 static void findPrimefactors(HashSet<Integer> s, int n)  
 {  
 // Print the number of 2s that divide n  
 while (n % 2 == 0)  
 {  
 s.add(2);  
 n = n / 2;  
 }  
  
 // n must be odd at this point. So we can skip one element  
 for (int i = 3; i <= Math.*sqrt*(n); i = i + 2)  
 {  
 // While i divides n, print i and divide n  
 while (n % i == 0)  
 {  
 s.add(i);  
 n = n / i;  
 }  
 }  
  
 // This condition is to handle the case when n is a prime number greater than 2  
 if (n > 2)  
 {  
 s.add(n);  
 }  
 }  
  
 // Function to find smallest primitive root of n  
 static int findPrimitive(int n)  
 {  
 HashSet<Integer> s = new HashSet<Integer>();  
  
 // Check if n is prime or not  
 if (*isPrime*(n) == false)  
 {  
 return -1;  
 }  
  
 // Find value of Euler Totient function of n  
 // Since n is a prime number, the value of Euler  
 // Totient function is n-1 as there are n-1  
 // relatively prime numbers.  
 int eul = n - 1;  
  
 // Find prime factors of eul and store in a set  
 *findPrimefactors*(s, eul);  
  
 // Check for every number from 2 to eul  
 for (int r = 2; r <= eul; r++)  
 {  
 // Iterate through all prime factors of eul.  
 // and check if we found a power with value 1  
 boolean flag = false;  
 for (Integer a : s)  
 {  
  
 // Check if r^((eul)/primefactors) mod n  
 // is 1 or not  
 if (*power*(r, eul / (a), n) == 1)  
 {  
 flag = true;  
 break;  
 }  
 }  
  
 // If there was no power with value 1.  
 if (flag == false)  
 {  
 return r;  
 }  
 }  
  
 // If no primitive root found  
 return -1;  
 }  
  
 // Driver code  
 public static void main(String[] args)  
 {  
 int n = 13;  
 System.*out*.println(" Smallest primitive root of " + n  
 + " is " + *findPrimitive*(n));  
 }  
}

Problem 2 output and code:

Text

Description automatically generated

class Mitm {  
 private static long power(long a, long b, long p) {  
 if (b == 1)  
 return a;  
 else  
 return (((long) Math.*pow*(a, b)) % p);  
 }  
  
 public static void main(String[] args) {  
 long P, G, x, a, y, b, z, c, w, d, ga, gb, gc, gd, S1, S2;  
  
 // A prime number P is taken  
 P = 13;  
 System.*out*.println("The value of P: " + P);  
  
 // A primitive root for P, G is taken  
 G = 6;  
 System.*out*.println("The value of G: " + G + "\n");  
  
 // Alice will choose the private key a  
 // a is the chosen private key  
 a = 5;  
 System.*out*.println("The private key a for Alice(a): " + a);  
  
 // Gets the generated key  
 x = *power*(G, a, P);  
  
 // Bob will choose the private key b  
 // b is the chosen private key  
 b = 2;  
 System.*out*.println("The private key b for Bob(b): " + b + "\n");  
  
 // Gets the generated key  
 y = *power*(G, b, P);  
  
  
 // Mallory will choose two random numbers (c)  
 c = 72;  
 System.*out*.println("Mallory selected private number for Alice(c): " + c);  
  
 // Gets the generated key  
 z = *power*(G, c, P);  
  
 // Mallory will choose two random numbers (d)  
 d = 134;  
 System.*out*.println("Mallory selected private number for Bob(d): " + d + "\n");  
  
 // Gets the generated key  
 w = *power*(G, d, P);  
  
 ga = *power*(y, a, P); // Secret key for Alice  
 gb = *power*(x, b, P); // Secret key for Bob  
 gc = *power*(w, c, P); // Secret key from mallory for Alice  
 gd = *power*(z, d, P); // Secret key from mallory for Bob  
  
 S1 = *power*(G, d, a)%P; // Alice  
 S2 = *power*(G, c, b)%P; // Bob  
  
 System.*out*.println("Secret key for Alice is(ga): " + ga);  
 System.*out*.println("Secret key for Bob is(gb): " + gb + "\n");  
 System.*out*.println("Eve published value for Alice (gc): " + gc);  
 System.*out*.println("Eve published value for Bob (gd): " + gd + "\n");  
 System.*out*.println("Alice computed (S1): " + S1);  
 System.*out*.println("Mallory computed key for Alice (S1): " + S1 + "\n");  
 System.*out*.println("Bob computed (S2): " + S2);  
 System.*out*.println("Mallory computed key for Bob (S2): " + S2);  
  
 }  
}